

Sliding Window Technique

The **sliding window** is originally a concept from **networking**, where it's used in protocols. A **protocol** is simply a set of rules that governs how data is transmitted between systems. The idea of the sliding window in programming is inspired by this networking principle.

In the context of **problem solving**, the sliding window is a powerful technique used to optimize solutions involving **arrays or strings**.

✅ When to Apply Sliding Window

You can apply the sliding window technique when:

- The input includes an **array or string**
- You're given an **integer k** (which usually represents window size)
- The output demands:
 - A **substring** or **subarray**
 - A condition based on **maximum, minimum, sum, count, or frequency**

The **sliding window technique comes in two forms**:

- A **fixed-size** window, where the window length remains constant throughout the traversal.
- A **variable-size** window, where the window expands or contracts dynamically based on a condition.



Maximum Sum of K Consecutive Elements

You are given an array of integers `arr[]` and an integer `k`. Your task is to find the maximum sum of any contiguous subarray of size `k`.



Brute Force Approach - $O(n*k)$

```
public class Main {
    public static void main(String[] args) {
        int[] arr = { 2, 3, 5, 1, 5, 7, 8, 9, 1 };
        int k = 3;
        System.out.println(maximumSumBruteForce(arr, k)); // Output: 24
    }

    public static int maximumSumBruteForce(int[] arr, int k) {
        int n = arr.length;
        int maxSum = Integer.MIN_VALUE;

        // Loop through all possible subarrays of size k
        for (int i = 0; i <= n - k; i++) {
            int currentSum = 0;

            // Calculate sum of subarray starting at i
            for (int j = i; j < i + k; j++) {
                currentSum += arr[j];
            }

            // Update maximum sum
            maxSum = Math.max(maxSum, currentSum);
        }
    }
}
```

```

    return maxSum;
}
}

```

🐰 Sliding Window approach - $O(n)$

maximum sum of k consecutive elements

arr \rightarrow [2, 3, 1, 5, 7, 9, 4, 3, 2] $k = 3$

$$\rightarrow 5 + 7 + 9 = 21$$

\rightarrow Find subarray of size k whose sum is max.

0	1	2	3	4	5	6	7	8
2	3	1	5	7	9	4	3	2

1st window \Rightarrow 2, 3, 1 = 6

max \rightarrow 5, 7, 9 = 21

$sum = 6 + 5 = 11 - 2 = 9 \Rightarrow$

	add	sub
$arr[3]$	$arr[0]$	
$arr[4]$	$arr[1]$	
$arr[5]$	$arr[2]$	
$arr[6]$	$arr[3]$	
$arr[7]$	$arr[4]$	
$arr[8]$	$arr[5]$	

$9 + 7 = 16 - 3 = 13$
 $13 + 9 = 22 - 1 = 21$
 $21 + 4 = 25 - 5 = 20$
 $20 + 3 = 23 - 7 = 16$
 $16 + 2 = 18 - 9 = 9$

\swarrow $arr[i], arr[i-k]$

sliding window of fixed size

1st window calculate

{
 // Grow
 // Shrink
 // Update my answer.
 }

```

public class Main {
    public static void main(String[] args) {
        int[] arr = { 2, 3, 5, 1, 5, 7, 8, 9, 1 };
        int k = 3;
        System.out.println(maxiumSum(arr, k));
    }

    public static int maxiumSum(int[] arr, int k) {
        int ans = 0;
        int sum = 0;

        // 1st window calculation
        for (int i = 0; i < k; i++) {
            sum += arr[i];
        }
        ans = sum;

        for (int i = k; i < arr.length; i++) {
            sum += arr[i]; // Grow
            sum -= arr[i - k]; // Shrink
            ans = Math.max(ans, sum);
        }
        return ans;
    }
}

```

🎯 Subarray Product Less Than K

<https://leetcode.com/problems/subarray-product-less-than-k/>

```

loop {
    // grow
    // shrink
    // update answer
}

```

arr \rightarrow 1, 2, 1, 3, 4, 2

k = 10

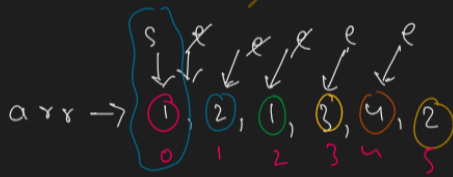
prod = ~~2~~ ~~4~~ ~~12~~ 4



1 2 1 3 4 2
 1 2 2 1 1 3 4 2
 1 2 1 2 1 3
 1 2 1 3

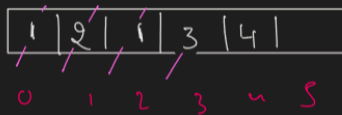
1 + 2 + 3 + 4 + 1 + 2 = 13

Grow \rightarrow Product
 Shrink \rightarrow Divide



k = 10

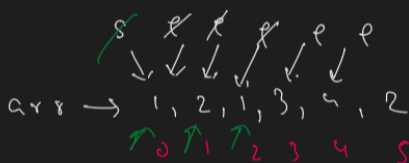
prod = ~~2~~ ~~4~~ ~~12~~ 4



start \rightarrow 0

end \rightarrow ~~2~~ ~~4~~ ~~12~~ 4

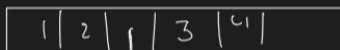
grow \rightarrow end++
 shrink \rightarrow start++



end = 3

start = 0

window = end - start + 1



```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        int[] arr = { 1, 2, 1, 3, 4, 2 };
```

```
        int k = 10;
```

```
        System.out.println(Product_Less_Than_K(arr, k));
```

```
    }
```

```
    public static int Product_Less_Than_K(int[] arr, int k) {
```

```
        int ans = 0;
```

```
        int end = 0, start = 0, prod = 1;
```

```

        while (end < arr.length) {
            // Grow
            prod = prod * arr[end];

            // Shrink
            while (prod >= k) {
                prod /= arr[start];
                start++;
            }

            // calculate ans
            ans += (end-start+1);

            end++;
        }
        return ans;
    }
}

```

```

class Solution {
    public int numSubarrayProductLessThanK(int[] nums, int k) {

        if (k <= 1) return 0;

        int ans = 0;
        int end = 0, start = 0, prod = 1;
        while (end < nums.length) {
            // Grow
            prod = prod * nums[end];

            // Shrink
            while (prod >= k && start<=end) {
                prod /= nums[start];
                start++;
            }

            // calculate ans
            ans += (end-start+1);

            end++;
        }
        return ans;
    }
}

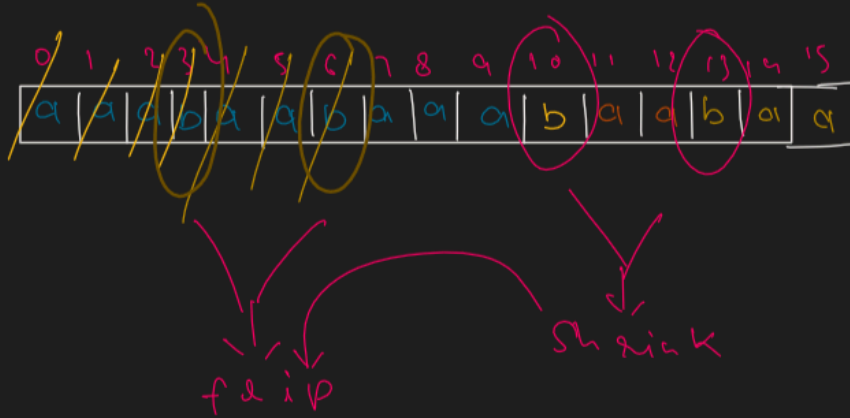
```

Kartik Bhaiya And Strings

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
a a a b a a b a a a b a a b a a

k = 2

maximum a \rightarrow max \Rightarrow ans
maximum b



s = 0
e = 0
flip = 0
max = 0

calculator
max

s = 0, e = 0

while (e < arr.length)

// ans = 0

// shrink

// Answer

update

}

```
import java.util.*;
```

```
public class Main {
```

```
    public static void main(String[] args) {
```

```

// Scanner sc = new Scanner(System.in);
// int k = sc.nextInt();
// String str = sc.next();

int k = 2;
String str = "abababbababaaaaaabbabbbaaa";
int max_a = max_length(str, 'b', k);
int max_b = max_length(str, 'a', k);
System.out.println(Math.max(max_a, max_b));
}

public static int max_length(String s, char ch, int k) {
    int si = 0, ei = 0, ans = 0, flip = 0;
    while (ei < s.length()) {
        // grow
        if (s.charAt(ei) == ch) {
            flip++;
        }
        // shrink
        while (flip > k && si <= ei) {
            if (s.charAt(si) == ch) {
                flip--;
            }
            si++;
        }
        // calculate ans
        ans = Math.max(ans, ei - si + 1);
        ei++;
    }
    return ans;
}
}

```

🎯 Arrays-Sum Of Two Arrays

Handwritten calculations showing the sum of two arrays:

Array 1: 3 2 9
Array 2: 4 7 6
Sum: 8 0 5

Array 1: 9 9 9
Array 2: 9 9 9
Sum: 1 9 9 8

Array 1: 9 9 9 5
Array 2: 9 9 9 7
Sum: 9 9 9 7

Array 1: 9 9 3 5
Array 2: 9 9 4 2
Sum: 9 9 4 2

Array 1: 9 9 9 9
Array 2: 1 0 0 0 0
Sum: 1 0 0 0 0

```

import java.util.*;
public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        int n1 = sc.nextInt();
        int[] arr1 = new int[n1];
        for(int i=0; i<n1; i++){
            arr1[i] = sc.nextInt();
        }
        int n2 = sc.nextInt();
        int[] arr2 = new int[n2];
        for(int i=0; i<n2; i++){
            arr2[i] = sc.nextInt();
        }
        // int[] arr1 = { 1, 0, 2, 9 };
        // int[] arr2 = { 3, 2, 4, 5, 6, 7 };
        int[] ans = sumOfTwoArrays(arr1, arr2);
        for (int i = 0; i < ans.length; i++) {
            System.out.print(ans[i] + ", ");
        }
        System.out.println("END");
    }

    public static int[] sumOfTwoArrays(int[] a1, int[] a2){
        ArrayList<Integer> list = new ArrayList<>();
        int i = a1.length - 1;
        int j = a2.length - 1;

        int carry = 0;
        while(i>=0 && j>=0){
            int sum = a1[i] + a2[j] + carry;
            if(sum>9){
                sum = sum%10;
                carry = 1;
            } else{
                carry = 0;
            }
            list.add(sum);
            i--;
            j--;
        }
        while(i>=0){
            int sum = a1[i] + carry;
            if(sum>9){
                sum = sum%10;
                carry = 1;
            } else{
                carry = 0;
            }
            list.add(sum);
            i--;
        }
    }
}

```



```
while(j>=0){
    int sum = a2[j] +carry;
    if(sum>9){
        sum = sum%10;
        carry = 1;
    } else{
        carry = 0;
    }
    list.add(sum);
    j--;
}

if(carry>0){
    list.add(carry);
}

int[] ans = new int[list.size()];
int l = 0;
for (int k = list.size() - 1; k >= 0; k--) {
    ans[l++] = list.get(k);
}
return ans;
}
}
```